Лабораторна робота "Фрагментарна реалізація систем управління табличними базами даних"

I. Загальні вимоги

Основні вимоги щодо структури бази:

- кількість таблиць принципово не обмежена (реляції між таблицями не враховувати);
- кількість полів та кількість записів у кожній таблиці також принципово не обмежені.

У кожній роботі треба забезпечити підтримку (для полів у таблицях) наступних (загальних для всіх варіантів!) типів:

- integer
- real
- char
- string

Також у кожній роботі треба реалізувати функціональну підтримку для:

- створення бази
- створення (із валідацією даних) та знищення таблиці з бази
- перегляду та редагування рядків таблиці
- збереження табличної бази на диску та, навпаки, зчитування її з диску

II. Варіанти додаткових типів

- color (RGB код кольору); colorInvl (інтервальний тип)

III. Варіанти додаткових операцій над таблицями

- вилучення повторюваних рядків у таблиці

V. Завдання лабораторного практикуму

- 0 Етап
- 1 Етап
- 2-3 Етап
- 10 Етап
- 11 Етап
- 12 Етап
- 13 Етап
- 14 Етап
- 20 Етап
- 24 Етап
- 27 Етап

Звіт до 0 етапу

UseCase.png - UML діаграма прецедентів, яка призначена для проектування та специфікації програмних систем.

Звіт до 1 етапу

12 UML діаграм до лабораторної роботи призначені для проектування та специфікації програмних систем:

- *Class diagram.png* містить діаграму класів. Містить класи "Table", "Database", "Column", також "AbstractRestController" та його видозміни ("TableManagementRestController", "ReadWriteTableRestController", "DataManagementRestController").
- *Components.png* містить діаграму компонентів для розподіленої системи.
- *Sequence*Create_from_file.png_ - діаграма послідовностей для створення бази даних з файлу.
- *Sequence_Edit_Value.png* - діаграма послідовностей для редагування значень в таблиці.
- *Sequence_deduplicate.png* - діаграма послідовностей для видалення повторюваних рядків.
- *Sequence_diagr_fetch.png* - діаграма послідовностей для витягування всіх таблиць.
- *Sequence_save_to_drive.png* - діаграма послідовностей для збереження бази даних у файл.
- *Sequence_table_create.png* - діаграма послідовностей для створення таблиці у базі даних.
- *UseCase.png* - діаграма прецедентів.
- *VOPC_Class.png* - VOPC-діаграма для створення таблиці.
- *VOPC_deduplication.png* - VOPC-діаграма для видалення повторюваних рядків у таблиці.

- *Deployment_diagram.png* - діаграма розгортання.
- *Flow_of_events.txt* - до трьох прецедентів (створення бази даних, вилучення повторюваних рядків, створення бази даних з файлу) описано потоки подій.

Звіт до 2-3 етапу

Було розроблено класи "Table", "Database" та "TableColumn" для понять "Таблиця", "База даних" та "Колонка таблиці" (папка models в структурі проєкту). Також було додано UML-діаграму класів *Class_diagram.png*. У папці test_dedup знаходяться три Unit-тести, для вилучення повторюваних рядків таблиці, зокрема для перевірки рядків на рівність та знаходження повторюваних рядків за ідентифікатором. Інтерфейс користувача забезпечується за допомогою реалізованої фронтенд-частини. Операції, реалізовані у фронтенд частині: - Створення БД - Створення таблиці - Додавання рядка - Дедуплікація рядків - Перегляд та редагування таблиці - Отримання даних з БД - Відтворення БД з даних

HTTP-запити протестовані у Postman.

Звіт до 10 етапу

REST web-сервіси.

REST web-сервіси реалізовані на фреймворку FastAPI. Ієрархічна структура має наступний вигляд: /database/{databaseId}/table/{tableId}. Реалізований REST API сервер, HTTP-запити протестовані у Postman.

Звіт до 11 етапу

REST web-сервіси + HATEOAS

REST web-сервіси реалізовані на фреймворку FastAPI. Ієрархічна структура має наступний вигляд: /database/{databaseId}/table/{tableId}. Реалізований REST API сервер, HTTP-запити протестовані у Postman.

Архітектурні обмеження для REST-додатків реалізовані за допомогою методу *hateoas_links()*, який повертає дозволені операції для бази даних та таблиці залежно від того, чи існують вони.

Звіт до 12 етапу

REST web-сервіси. Розробка OpenAPI Specification для взаємодії з ієрархічними даними (база, таблиця, ...).

REST web-сервіси реалізовані на фреймворку FastAPI. Ієрархічна структура має наступний вигляд: /database/{databaseId}/table/{tableId}. Реалізований REST API сервер, HTTP-запити протестовані у Postman.

Файл openapispec.yaml був написаний використовуючи програму OpenAPI Generator.

Звіт до 13 етапу

REST web-сервіси. Реалізація серверного проєкту, використовуючи кодогенерацію стабу за OpenAPI Specification.

REST web-сервіси реалізовані на фреймворку FastAPI. Ієрархічна структура має наступний вигляд: /database/{databaseId}/table/{tableId}. Реалізований REST API сервер, HTTP-запити протестовані у Postman.

Файл openapispec.yaml був написаний використовуючи програму OpenAPI Generator. Використовуючи інтерфейс OpenAPI, було згенеровано загальну структуру проекту.

Звіт до 14 етапу

REST web-сервіси. Реалізація клієнтського проєкту за OpenAPI Specification.

REST web-сервіси реалізовані на фреймворку FastAPI. Ієрархічна структура має наступний вигляд: /database/{databaseId}/table/{tableId}. Реалізований REST API сервер, HTTP-запити протестовані у Postman.

Файл openapispec.yaml був написаний використовуючи програму OpenAPI Generator. Клієнтський проект - це бібліотека, яку використовується в додатку, що працює на API, тобто, робить запити до API, описані у визначенні OpenAPI.

Звіт до 20 етапу

Web-проект із використанням AJAX. Щонайменше повинно забезпечуватись часткове перезавантаження web-сторінки.

В основі підходу до побудови користувацьких інтерфейсів AJAX покладена ідея надсилання веб-сторінкою запитів на сервер у фоновому режимі і довантажування користувачу необхідних даних.

Звіт до 24 етапу

Варіант проєкту із використанням реляційної СУБД (замість використання серіалізації об'єктів для збереження даних).

Проєкт реалізований з використанням класів "Таблиця" (table.py), "Колонка" (table_columns.py), "База даних" (database.py), з відношенням "один до багатьох". Уси три файли розташовані у папці проєкту *models*. Функціонал був реалізований на сонові попередньо створених Use-Case діаграм.

Звіт до 27 етапу

Інтегроване (Mock-) тестування у проєктах, що використовують реальні СУБД (реляційні чи ні) для збереження даних.

Реалізовані інтеграційні тестування: - додавання рядка (*test_add_row()*) - створення бази даних (*test_database_create_post()*) - створення однакової бази даних (*test_same_database_creation()*) - створення таблиці (*test_create_table_for_existing_db()*) - видалення повторюваних рядків таблиці (*test_dedup()*) - видалення таблиці (*test_delete_table()*) - редагування рядка таблиці (_test_edit_value()) - редагування інтервалу інтервалу кольорів (test_edit_value_range()) - редагування значення кольору (test_edit_value_color())

За успішного реалізації повертається код 201.